####笔记
韩顺平.循序渐进学java一笔记
Java核心技术_笔记
深入理解Java虚拟机一笔记

==========updating==========

####
Java 序列化的高级认识
1.序列化ID的问题：虚拟机是否允许反序列化，不仅取决于类路径和功能代码是否一致，一个非常重要的一点是两个类的序列化 ID 是否一致（就是 private static final long serialVersionUID = 1L）。清单 1 中，虽然两个类的功能代码完全一致，但是序列化 ID 不同，他们无法相互序列化和反序列化。
序列化 ID 在 Eclipse 下提供了两种生成策略，一个是固定的 1L，一个是随机生成一个不重复的 long 类型数据（实际上是使用 JDK 工具生成），在这里有一个建议，如果没有特殊需求，就是用默认的 1L 就可以，这样可以确保代码一致时反序列化成功。那么随机生成的序列化 ID 有什么作用呢，有些时候，通过改变序列化 ID 可以用来限制某些用户的使用。
2.静态变量序列化：序列化保存的是对象的状态，静态变量属于类的状态，因此 序列化并不保存静态变量。
3.父类的序列化与 Transient 关键字：要想将父类对象也序列化，就需要让父类也实现Serializable 接口。如果父类不实现的话的，就 需要有默认的无参的构造函数。在父类没有实现 Serializable 接口时，虚拟机是不会序列化父对象的，而一个 Java 对象的构造必须先有父对象，才有子对象，反序列化也不例外。所以反序列化时，为了构造父对象，只能调用父类的无参构造函数作为默认的父对象。因此当我们取父对象的变量值时，它的值是调用父类无参构造函数后的值。如果你考虑到这种序列化的情况，在父类无参构造函数中对变量进行初始化，否则的话，父类变量值都是默认声明的值，如 int 型的默认是 0，string 型的默认是 null。
Transient 关键字的作用是控制变量的序列化，在变量声明前加上该关键字，可以阻止该变量被序列化到文件中，在被反序列化后，transient 变量的值被设为初始值，如 int 型的是 0，对象型的是 null。
4.对敏感字段加密：在序列化过程中，虚拟机会试图调用对象类里的 writeObject 和 readObject 方法，进行用户自定义的序列化和反序列化，如果没有这样的方法，则默认调用是 ObjectOutputStream 的 defaultWriteObject 方法以及 ObjectInputStream 的 defaultReadObject 方法。用户自定义的 writeObject 和 readObject 方法可以允许用户控制序列化的过程，比如可以在序列化的过程中动态改变序列化的数值。基于这个原理，可以在实际应用中得到使用，用于敏感字段的加密工作，
5.序列化存储规则：Java 序列化机制为了节省磁盘空间，具有特定的存储规则，当写入文件的为同一对象时，并不会再将对象的内容进行存储，而只是再次存储一份引用，上面增加的 5 字节的存储空间就是新增引用和一些控制信息的空间。反序列化时，恢复引用关系，使得清单 3 中的 t1 和 t2 指向唯一的对象，二者相等，输出 true。该存储规则极大的节省了存储空间。
原本是希望一次性传输对象修改前后的状态，写入一次以后修改对象属性值再次保存第二次。第一次写入对象以后，第二次再试图写的时候，虚拟机根据引用关系知道已经有一个相同对象已经写入文件，因此只保存第二次写的引用，所以读取时，都是第一次保存的对象。读者在使用一个文件多次 writeObject 需要特别注意这个问题。

####
重写、覆盖、重载、多态几个概念的区别分析 - Bolt 的专栏 - 博客频道 - CSDN.NET
override->重写(=覆盖)、overload->重载、polymorphism -> 多态
override是重写（覆盖）了一个方法，以实现不同的功能。一般是用于子类在继承父类时，重写（重新实现）父类中的方法。
重写（覆盖）的规则：
1、重写方法的参数列表必须完全与被重写的方法的相同,否则不能称其为重写而是重载.
2、重写方法的访问修饰符一定要大于被重写方法的访问修饰符（public>protected>default>private）。
3、重写的方法的返回值必须和被重写的方法的返回一致；
4、重写的方法所抛出的异常必须和被重写方法的所抛出的异常一致，或者是其子类；
5、被重写的方法不能为private，否则在其子类中只是新定义了一个方法，并没有对其进行重写。
6、静态方法不能被重写为非静态的方法（会编译出错）。
overload是重载，一般是用于在一个类内实现若干重载的方法，这些方法的名称相同而参数形式不同。
重载的规则：
1、在使用重载时只能通过相同的方法名、不同的参数形式实现。不同的参数类型可以是不同的参数类型，不同的参数个数，不同的参数顺序（参数类型必须不一样）；
2、不能通过访问权限、返回类型、抛出的异常进行重载；
3、方法的异常类型和数目不会对重载造成影响；
多态的概念比较复杂，有多种意义的多态，一个有趣但不严谨的说法是：继承是子类使用父类的方法，而多态则是父类使用子类的方法。

####
Thread_wait、notify、notifyAll的使用方法 - qaz13177_58_的专栏 - 博客频道 - CSDN.NET
1.线程取得控制权的方法有三：1.执行对象的某个同步实例方法。2.执行对象对应类的同步静态方法。3.执行对该对象加同步锁的同步块。
2.如果对在同步块中对flag进行了赋值操作，使得flag引用的对象改变，这时候再调用notify方法时，因为没有控制权所以抛出异常
3.调用一个Object的wait与notify/notifyAll的时候，必须保证调用代码对该Object是同步的，也就是说必须在作用等同于synchronized(obj){.....}的内部才能够去调用obj的wait与notify/notifyAll三个方法，否则就会报错：java.lang.IllegalMonitorStateException:current thread not owner
4.wait(),notify(),notifyAll()不属于Thread类,而是属于Object基础类,也就是说每个对像都有wait(),notify(),notifyAll()的功能。因为都个对像都有锁,锁是每个对像的基础,当然操作锁的方法也是最基础了。
5.#调用obj的wait(), notify()方法前，必须获得obj锁，也就是必须写在synchronized(obj){...} 代码段内。
调用obj.wait()后，线程A就释放了obj的锁，否则线程B无法获得obj锁，也就无法在synchronized(obj){...} 代码段内唤醒A。
当obj.wait()方法返回后，线程A需要再次获得obj锁，才能继续执行。

####
Java的运行原理 - java_andy - 博客园
1.Java源程序经过编译器编译后变成字节码，字节码由虚拟机解释执行，虚拟机将每一条要执行的字节码送给解释器，解释器将其翻译成特定机器上的机器码，然后在特定的机器上运行。
2.Java中，类加载器把一个类装入JAVA虚拟机需要经过三个步骤来完成：装载、链接、初始化，其中链接又分来校验、准备、解析过程
装载：查找和导入.class文件
链接：检查装入.class文件的正确性，然后，java虚拟机为变量分配内存，设置默认值
初始化：把符号引用变成直接引用。。。

####
java多线程总结 - Rollen Holt - 博客园
1.为什么我们不能直接调用run()方法呢？因为：线程的运行需要本地操作系统的支持。查看start源码，会调用到native方法中。
2.但是start方法重复调用的话，会出现java.lang.IllegalThreadStateException异常。
3.Thread也是实现Runnable接口的，其实Thread中的run方法调用的是Runnable接口的run方法。Thread和Runnable都实现了run方法，这种操作模式其实就是代理模式。
3.如果一个类继承Thread，则不适合资源共享。但是如果实现了Runable接口的话，则很容易的实现资源共享。
总结实现Runnable接口比继承Thread类所具有的优势：
a: 适合多个相同的程序代码的线程去处理同一个资源
b: 可以避免java中的单继承的限制
c:增加程序的健壮性，代码可以被多个线程共享，代码和数据独立。
4.main方法其实也是一个线程。在java中所以的线程都是同时启动的，至于什么时候，哪个先执行，完全看谁先得到CPU的资源。在java中，每次程序运行至少启动2个线程。一个是main线程，一个是垃圾收集线程。因为每当使用java命令执行一个类的时候，实际上都会启动一个ＪＶＭ，每一个ｊＶＭ实习在就是在操作系统中启动了一个进程。
5.在java程序中，只要前台有一个线程在运行，整个java程序进程不会结束。主线程也有可能在子线程结束之前结束，子线程不受影响，不会因为主线程的结束而结束。 但是，调用demo.setDaemon(true);设置为后台线程后，如果主线程结束，整个进程也就结束了，意味着后台线程也会结束。
6.demo.setPriority(8);设置线程的优先级，主线程的优先级默认是5.
7.在线程操作中，也可以使用yield()方法，将一个线程的操作暂时交给其他线程执行。
8.需要同步的话，可以使用同步代码块和同步方法两种来完成。synchronized
9.使用Object的wait、notify方法解决生产者消费者数据消费的同步问题。

####
java 匿名类（转载） - henry_xu - 博客园

####match方法，
```
1517    @Override
1518    public boolean isMEMCDemoModeSupported() {
1519        return !SystemProperties.get("ro.product.family", "").matches("X4[MNO]");
1520    }
```

####instanceof关键字用于判断一个引用类型变量所指向的对象是否是一个类（或接口、抽象类、父类）的实例

####eclipse
ctrl+/　　注释
ctrl+shift+f　对齐